

Karlsruhe Reports in Informatics 2010,11

Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Dynamic Frames in Java Dynamic Logic Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, Benjamin Weiß

2010



Fakultät für Informatik

Please note:

This Report has been published on the Internet under the following
Creative Commons License:

<http://creativecommons.org/licenses/by-nc-nd/3.0/de>.

Dynamic Frames in Java Dynamic Logic

Formalisation and Proofs

Peter H. Schmitt, Mattias Ulbrich, and Benjamin Weiß

Karlsruhe Institute of Technology
Institute for Theoretical Computer Science
D-76128 Karlsruhe, Germany
{pschmitt,mulbrich,bweiss}@ira.uka.de

Abstract. This report is a companion to the paper *Dynamic Frames in Java Dynamic Logic* [2]. It contains complementary formal definitions and proofs.

1 Formalisation

1.1 Syntax

Definition 1 (Signatures). A signature Σ is a tuple

$$\Sigma = (\mathcal{T}, \sqsubseteq, \mathcal{V}, \mathcal{PV}, \mathcal{F}, \mathcal{P}, \alpha, \text{Prg})$$

where \mathcal{T} is a finite set of types; where \sqsubseteq is a partial order on \mathcal{T} called the subtype relation; where \mathcal{V} is a set of (logical) variables; where \mathcal{PV} is a set of program variables; where \mathcal{F} is a set of function symbols; where \mathcal{P} is a set of predicate symbols; where α is a static typing function such that $\alpha(v) \in \mathcal{T}$ for all $v \in \mathcal{V} \cup \mathcal{PV}$, $\alpha(f) \in \mathcal{T}^* \times \mathcal{T}$ for all $f \in \mathcal{F}$, and $\alpha(p) \in \mathcal{T}^*$ for all $p \in \mathcal{P}$; and where Prg is some Java program, i.e., a set of Java classes and interfaces.

We use the notation $v:A$ for $\alpha(v) = A$, the notation $f:A_1, \dots, A_n \rightarrow A$ for $\alpha(f) = ((A_1, \dots, A_n), A)$, and the notation $p:A_1, \dots, A_n$ for $\alpha(p) = (A_1, \dots, A_n)$.

We require that the following types, program variables, function and predicate symbols are present in every signature:

- $\text{Any}, \text{Boolean}, \text{Int}, \text{Null}, \text{LocSet}, \text{Field}, \text{Heap} \in \mathcal{T}$
- all reference types of Prg also appear as types in \mathcal{T} ; in particular, $\text{Object} \in \mathcal{T}$
- all local variables \mathbf{a} of Prg with Java type T also appear as program variables $\mathbf{a}:A \in \mathcal{PV}$, where $A = T$ if T is a reference type, $A = \text{Boolean}$ if $T = \text{boolean}$, and $A = \text{Int}$ if $T = \text{int}$ (in this paper we do not consider other primitive types, and we ignore integer overflows)
- $\text{heap}: \text{Heap} \in \mathcal{PV}$
- $\text{cast}_A: \text{Any} \rightarrow A \in \mathcal{F}$ (for every type $A \in \mathcal{T}$)
- $\text{TRUE}, \text{FALSE}: \text{Boolean} \in \mathcal{F}$
- $\text{select}_A: \text{Heap}, \text{Object}, \text{Field} \rightarrow A \in \mathcal{F}$ (for every type $A \in \mathcal{T}$)

- $store : Heap, Object, Field, Any \rightarrow Heap \in \mathcal{F}$
- $anon : Heap, LocSet, Heap \rightarrow Heap \in \mathcal{F}$
- $null : Null \in \mathcal{F}$
- all Java fields f of Prg also appear as constant symbols $f : Field \in \mathcal{F}$
- $arr : Int \rightarrow Field \in \mathcal{F}$, $created : Field \in \mathcal{F}$
- $allLocs : LocSet \in \mathcal{F}$, $allFields : Object \rightarrow LocSet \in \mathcal{F}$, $freshLocs : Heap \rightarrow LocSet \in \mathcal{F}$
- $\emptyset : LocSet \in \mathcal{F}$, $singleton : Object, Field \rightarrow LocSet \in \mathcal{F}$
- $\dot{\cup}, \dot{\cap}, \setminus : LocSet, LocSet \rightarrow LocSet \in \mathcal{F}$
- $exactInstance_A : Any \in \mathcal{P}$ (for every type $A \in \mathcal{T}$)
- $wellFormed : Heap \in \mathcal{P}$
- $\dot{=} : Any, Any \in \mathcal{P}$
- $\dot{\in} : Object, Field, LocSet \in \mathcal{P}$, $\dot{\subseteq}, disjoint : LocSet, LocSet \in \mathcal{P}$

We also require that $Boolean, Int, Object, LocSet \sqsubseteq Any$; that for all $C \in \mathcal{T}$ with $C \sqsubseteq Object$ we have $Null \sqsubseteq C$; that for all types A, A' of Prg we have $A' \sqsubseteq A$ if and only if A' is a subtype of A in Prg ; that the types explicitly mentioned in this definition are otherwise unrelated to each other wrt. \sqsubseteq ; and that the types $Boolean, Int, Null, LocSet, Field$ and $Heap$ do not have subtypes except themselves. Finally, we demand that $\mathcal{V}, \mathcal{PV}, \mathcal{F}$ and \mathcal{P} each contain an infinite number of symbols of every typing.

For illustration, the type hierarchy is visualised in Fig. 1. In the following, we assume a fixed signature $\Sigma = (\mathcal{T}, \sqsubseteq, \mathcal{V}, \mathcal{PV}, \mathcal{F}, \mathcal{P}, \alpha, Prg)$.

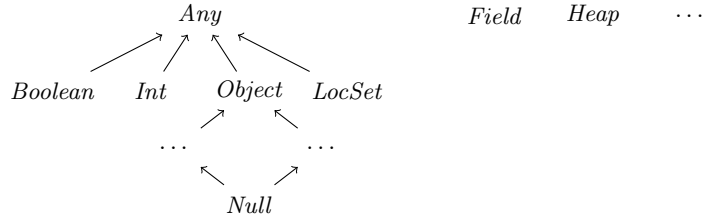


Fig. 1. Type hierarchy

Definition 2 (Syntax). The sets Trm_{Σ}^A of terms of type A , Fma_{Σ} of formulas and Upd_{Σ} of updates are defined by the following grammar:

$$\begin{aligned}
 Trm_{\Sigma}^A &::= x \mid \mathbf{a} \mid f(Trm_{\Sigma}^{B'_1}, \dots, Trm_{\Sigma}^{B'_n}) \mid \text{if}(Fma_{\Sigma})\text{then}(Trm_{\Sigma}^A)\text{else}(Trm_{\Sigma}^A) \mid \\
 &\quad \{Upd_{\Sigma}\} Trm_{\Sigma}^A \\
 Fma_{\Sigma} &::= true \mid false \mid p(Trm_{\Sigma}^{B'_1}, \dots, Trm_{\Sigma}^{B'_n}) \mid \neg Fma_{\Sigma} \mid Fma_{\Sigma} \wedge Fma_{\Sigma} \mid \\
 &\quad Fma_{\Sigma} \vee Fma_{\Sigma} \mid Fma_{\Sigma} \rightarrow Fma_{\Sigma} \mid Fma_{\Sigma} \leftrightarrow Fma_{\Sigma} \mid
 \end{aligned}$$

$$\begin{aligned} & \forall Ax; Fma_\Sigma \mid \exists Ax; Fma_\Sigma \mid [p]Fma_\Sigma \mid \langle p \rangle Fma_\Sigma \mid \{Upd_\Sigma\}Fma_\Sigma \\ Upd_\Sigma ::= & \mathbf{a} := Trm_\Sigma^{A'} \mid Upd_\Sigma \parallel Upd_\Sigma \mid \{Upd_\Sigma\}Upd_\Sigma \end{aligned}$$

for any variable $x : A \in \mathcal{V}$, any program variable $\mathbf{a} : A \in \mathcal{PV}$, any function symbol $f : B_1, \dots, B_n \rightarrow A \in \mathcal{F}$ and any predicate symbol $p : B_1, \dots, B_n$ where $B'_1 \sqsubseteq B_1, \dots, B'_n \sqsubseteq B_n$, any executable Java fragment \mathbf{p} , and any type $A' \in \mathcal{T}$ with $A' \sqsubseteq A$.

A sequent is a syntactical construct $\Gamma \Rightarrow \Delta$, where $\Gamma, \Delta \in 2^{Fma_\Sigma}$ are finite sets of formulas.

We use infix notation for the binary symbols $\dot{\cup}$, $\dot{\cap}$, $\dot{\setminus}$, $\dot{=}$, and $\dot{\subseteq}$. Furthermore, we use the notation $(A)t$ for $cast_A(t)$, the notation $o.f$ for $select_A(\mathbf{heap}, o, f)$ where $f : Field \in \mathcal{F}$ is a Java field, the notation $a[i]$ for $select_A(\mathbf{heap}, a, arr(i))$, the notation $o.*$ for $allFields(o)$, the notation $\{(o, f)\}$ for $singleton(o, f)$, the notation $t_1 \neq t_2$ for $\neg(t_1 \dot{=} t_2)$, the notation $(o, f) \dot{\in} s$ for $\dot{\in}(o, f, s)$, and the notation $(o, f) \dot{\notin} s$ for $\neg(o, f) \dot{\in}(s)$.

1.2 Semantics

Definition 3 (Kripke structures). A Kripke structure \mathcal{K} for a signature Σ is a tuple

$$\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$$

where \mathcal{D} is a set of semantical values called the domain; where δ is a dynamic typing function $\delta : \mathcal{D} \rightarrow \mathcal{T}$; where (using the definition $\mathcal{D}^A = \{d \in \mathcal{D} \mid \delta(d) \sqsubseteq A\}$) I is an interpretation function that maps every function symbol $f : A_1, \dots, A_n \rightarrow A \in \mathcal{F}$ to a function $I(f) : \mathcal{D}^{A_1}, \dots, \mathcal{D}^{A_n} \rightarrow \mathcal{D}^A$ and every predicate symbol $p : A_1, \dots, A_n \in \mathcal{P}$ to a relation $I(p) \subseteq \mathcal{D}^{A_1} \times \dots \times \mathcal{D}^{A_n}$; where \mathcal{S} is the set of all states, which are functions $s \in \mathcal{S}$ mapping every program variable $\mathbf{a} : A \in \mathcal{PV}$ to a value $s(\mathbf{a}) \in \mathcal{D}^A$; and where ρ is a function associating with every executable Java fragment \mathbf{p} in the context of Prg a transition relation $\rho(\mathbf{p}) \subseteq \mathcal{S}^2$ such that $(s_1, s_2) \in \rho(\mathbf{p})$ iff \mathbf{p} , when started in s_1 , terminates normally in s_2 (according to the Java semantics [1]). We consider Java programs to be deterministic, so for all program fragments \mathbf{p} and all $s_1 \in \mathcal{S}$, there is at most one s_2 such that $(s_1, s_2) \in \rho(\mathbf{p})$.

We require that every Kripke structure satisfies the following:

- $\mathcal{D}^{Boolean} = \{tt, ff\}$, $\mathcal{D}^{Int} = \mathbb{Z}$, $\mathcal{D}^{Null} = \{I(\mathbf{null})\}$, $\mathcal{D}^{LocSet} = 2^{\mathcal{D}^{Object} \times \mathcal{D}^{Field}}$, $\mathcal{D}^{Heap} = \mathcal{D}^{Object} \times \mathcal{D}^{Field} \rightarrow \mathcal{D}^{Any}$
- $\delta(d) \neq T$ for all $d \in \mathcal{D}$, if $T \in \mathcal{T}$ represents an interface or an abstract class
- $\{d \in \mathcal{D} \mid \delta(d) = T\}$ is infinite for all $T \sqsubseteq Object$, $T \neq Null$ not representing an interface or an abstract class
- $I(cast_A)(d) = d$ for all $d \in \mathcal{D}^A$
- $I(TRUE) = tt$, $I(FALSE) = ff$
- $I(select_A)(h, o, f) = I(cast_A)(h(o, f))$ for all $h \in \mathcal{D}^{Heap}$, $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$

- $I(\text{store})(h, o, f, d)(o', f') = \begin{cases} d & \text{if } o = o' \text{ and } f = f' \\ h(o', f') & \text{otherwise} \end{cases}$
for all $h \in \mathcal{D}^{\text{Heap}}, o, o' \in \mathcal{D}^{\text{Object}}, f, f' \in \mathcal{D}^{\text{Field}}, d \in \mathcal{D}^{\text{Any}}$
- $I(\text{anon})(h, s, h')(o, f) = \begin{cases} h'(o, f) & \text{if } ((o, f) \in s \text{ and } f \neq I(\text{created})) \\ & \text{or } (o, f) \in I(\text{freshLocs})(h) \\ h(o, f) & \text{otherwise} \end{cases}$
for all $h, h' \in \mathcal{D}^{\text{Heap}}, s \in \mathcal{D}^{\text{LocSet}}, o \in \mathcal{D}^{\text{Object}}, f \in \mathcal{D}^{\text{Field}}$
- let $\text{UniqueFunctions} \subseteq \mathcal{F}$ be the set consisting of the constant symbols representing Java fields, of `arr` and of `created`; then we require that for all $f, g \in \text{UniqueFunctions}$ the function $I(f)$ is injective, and that the ranges of the functions $I(f)$ and $I(g)$ are disjoint.
- $I(\text{allLocs}) = \mathcal{D}^{\text{Object}} \times \mathcal{D}^{\text{Field}}, I(\text{allFields})(o) = \{(o, f) \mid f \in \mathcal{D}^{\text{Field}}\},$
 $I(\text{freshLocs})(h) = \{(o, f) \in I(\text{allLocs}) \mid o \neq I(\text{null}), h(o, I(\text{created})) = \text{ff}\}$
- $I(\emptyset) = \emptyset, I(\text{singleton})(o, f) = \{(o, f)\}, I(\dot{\cup}) = \cup, I(\dot{\cap}) = \cap, I(\setminus) = \setminus$
- $I(\text{exactInstance}_A) = \{d \in \mathcal{D} \mid \delta(d) = A\}$
- $I(\text{wellFormed}) = \{h \in \mathcal{D}^{\text{Heap}} \mid \text{for all } o \in \mathcal{D}^{\text{Object}}, f \in \mathcal{D}^{\text{Field}}:$
 $\text{if } h(o, f) \in \mathcal{D}^{\text{Object}}, \text{ then } h(o, f) = I(\text{null})$
 $\text{or } h(h(o, f), I(\text{created})) = \text{tt}\}$
- $I(\dot{=}) = \{(d, d) \in \mathcal{D}^2\}$
- $I(\dot{\subseteq}) = \{(o, f, s) \in \mathcal{D}^{\text{Object}} \times \mathcal{D}^{\text{Field}} \times \mathcal{D}^{\text{LocSet}} \mid (o, f) \in s\}, I(\dot{\subseteq}) = \{(s_1, s_2) \in$
 $(\mathcal{D}^{\text{LocSet}})^2 \mid s_1 \subseteq s_2\}, I(\text{disjoint}) = \{(s_1, s_2) \in (\mathcal{D}^{\text{LocSet}})^2 \mid s_1 \cap s_2 = \emptyset\}$

Definition 4 (Semantics). Given a Kripke structure $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$, a state $s \in \mathcal{S}$ and a variable assignment $\beta : \mathcal{V} \rightarrow \mathcal{D}$ (where for every $x : A \in \mathcal{V}$ we have $\beta(x) \in \mathcal{D}^A$), we evaluate terms $t \in \text{Trm}_\Sigma^A$ to a value $\text{val}_{\mathcal{K}, s, \beta}(t) \in \mathcal{D}^A$, formulas $\varphi \in \text{Fma}_\Sigma$ to a truth value $\text{val}_{\mathcal{K}, s, \beta}(\varphi) \in \{\text{tt}, \text{ff}\}$, and updates $u \in \text{Upd}_\Sigma$ to a state transformer $\text{val}_{\mathcal{K}, s, \beta}(u) : \mathcal{S} \rightarrow \mathcal{S}$ as defined below.

$$\begin{aligned}
\text{val}_{\mathcal{K}, s, \beta}(x) &= \beta(x) \\
\text{val}_{\mathcal{K}, s, \beta}(\mathbf{a}) &= s(\mathbf{a}) \\
\text{val}_{\mathcal{K}, s, \beta}(f(t_1, \dots, t_n)) &= I(f)(\text{val}_{\mathcal{K}, s, \beta}(t_1), \dots, \text{val}_{\mathcal{K}, s, \beta}(t_n)) \\
\text{val}_{\mathcal{K}, s, \beta}(\text{if}(\varphi)\text{then}(t_1)\text{else}(t_2)) &= \begin{cases} \text{val}_{\mathcal{K}, s, \beta}(t_1) & \text{if } \text{val}_{\mathcal{K}, s, \beta}(\varphi) = \text{tt} \\ \text{val}_{\mathcal{K}, s, \beta}(t_2) & \text{otherwise} \end{cases} \\
\text{val}_{\mathcal{K}, s, \beta}(\{u\}t) &= \text{val}_{\mathcal{K}, s', \beta}(t), \text{ where } s' = \text{val}_{\mathcal{K}, s, \beta}(u)(s) \\
\text{val}_{\mathcal{K}, s, \beta}(\text{true}) &= \text{tt} \\
\text{val}_{\mathcal{K}, s, \beta}(\text{false}) &= \text{ff} \\
\text{val}_{\mathcal{K}, s, \beta}(p(t_1, \dots, t_n)) &= \text{tt} \text{ iff } (\text{val}_{\mathcal{K}, s, \beta}(t_1), \dots, \text{val}_{\mathcal{K}, s, \beta}(t_n)) \in I(p) \\
\text{val}_{\mathcal{K}, s, \beta}(\neg\varphi) &= \text{tt} \text{ iff } \text{val}_{\mathcal{K}, s, \beta}(\varphi) = \text{ff} \\
\text{val}_{\mathcal{K}, s, \beta}(\varphi_1 \wedge \varphi_2) &= \text{tt} \text{ iff } \text{ff} \notin \{\text{val}_{\mathcal{K}, s, \beta}(\varphi_1), \text{val}_{\mathcal{K}, s, \beta}(\varphi_2)\} \\
\text{val}_{\mathcal{K}, s, \beta}(\varphi_1 \vee \varphi_2) &= \text{tt} \text{ iff } \text{tt} \in \{\text{val}_{\mathcal{K}, s, \beta}(\varphi_1), \text{val}_{\mathcal{K}, s, \beta}(\varphi_2)\} \\
\text{val}_{\mathcal{K}, s, \beta}(\varphi_1 \rightarrow \varphi_2) &= \text{val}_{\mathcal{K}, s, \beta}(\neg\varphi_1 \vee \varphi_2) \\
\text{val}_{\mathcal{K}, s, \beta}(\varphi_1 \leftrightarrow \varphi_2) &= \text{val}_{\mathcal{K}, s, \beta}(\varphi_1 \rightarrow \varphi_2 \wedge \varphi_2 \rightarrow \varphi_1)
\end{aligned}$$

$$\begin{aligned}
val_{\mathcal{K},s,\beta}(\forall A x; \varphi) &= tt \text{ iff } ff \notin \{val_{\mathcal{K},s,\beta_x^d}(\varphi) \mid d \in \mathcal{D}^A\} \\
val_{\mathcal{K},s,\beta}(\exists A x; \varphi) &= tt \text{ iff } tt \in \{val_{\mathcal{K},s,\beta_x^d}(\varphi) \mid d \in \mathcal{D}^A\} \\
val_{\mathcal{K},s,\beta}([\mathbf{p}]\varphi) &= tt \text{ iff } ff \notin \{val_{\mathcal{K},s',\beta}(\varphi) \mid (s, s') \in \rho(\mathbf{p})\} \\
val_{\mathcal{K},s,\beta}(\langle \mathbf{p} \rangle \varphi) &= tt \text{ iff } tt \in \{val_{\mathcal{K},s',\beta}(\varphi) \mid (s, s') \in \rho(\mathbf{p})\} \\
val_{\mathcal{K},s,\beta}(\{u\}\varphi) &= val_{\mathcal{K},s',\beta}(\varphi), \text{ where } s' = val_{\mathcal{K},s,\beta}(u)(s) \\
val_{\mathcal{K},s,\beta}(\mathbf{a} := t)(s')(\mathbf{b}) &= \begin{cases} val_{\mathcal{K},s,\beta}(t) & \text{if } \mathbf{b} = \mathbf{a} \\ s'(\mathbf{b}) & \text{otherwise} \end{cases} \\
&\text{for all } s' \in \mathcal{S}, \mathbf{b} \in \mathcal{PV} \\
val_{\mathcal{K},s,\beta}(u_1 \parallel u_2)(s') &= val_{\mathcal{K},s,\beta}(u_2)(val_{\mathcal{K},s,\beta}(u_1)(s')) \text{ for all } s' \in \mathcal{S} \\
val_{\mathcal{K},s,\beta}(\{u_1\}u_2) &= val_{\mathcal{K},s',\beta}(u_2), \text{ where } s' = val_{\mathcal{K},s,\beta}(u_1)(s)
\end{aligned}$$

We sometimes write $(\mathcal{K}, s, \beta) \models \varphi$ instead of $val_{\mathcal{K},s,\beta}(\varphi) = tt$. A formula $\varphi \in Fma_{\Sigma}$ is called *logically valid*, in symbols $\models \varphi$, iff $(\mathcal{K}, s, \beta) \models \varphi$ for all Kripke structures \mathcal{K} , all states $s \in \mathcal{S}$, and all variable assignments β .

The semantics of a sequent $\Gamma \Rightarrow \Delta$ is the same as that of a formula $\bigwedge \Gamma \rightarrow \bigvee \Delta$, where $\bigvee \{\varphi_1, \dots, \varphi_n\} = \varphi_1 \vee \dots \vee \varphi_n$, and $\bigwedge \{\varphi_1, \dots, \varphi_n\} = \varphi_1 \wedge \dots \wedge \varphi_n$.

1.3 Observations

The propositions below are used as assumptions in the proofs in Sect. 2. We do not prove them, but consider them obvious.

Proposition 1 (Non-occurring program variables). *For all Kripke structures \mathcal{K} , all states $s, s' \in \mathcal{S}$, all variable assignments β , and all $t \in Trm_{\Sigma} \cup Fma_{\Sigma} \cup Upd_{\Sigma}$: if for all program variables $\mathbf{a} \in \mathcal{PV}$ that syntactically occur in t we have $s(\mathbf{a}) = s'(\mathbf{a})$, then we also have $val_{\mathcal{K},s,\beta}(t) = val_{\mathcal{K},s',\beta}(t)$.*

Note that a program variable \mathbf{b} not occurring in t can play a role in evaluating t , namely if t contains a program which calls a method that in turn manipulates \mathbf{b} . Still, in a Java program a called method can never read the value of a local variable \mathbf{b} before assigning to \mathbf{b} ; thus, the initial value of \mathbf{b} as defined by s or s' does not matter. We consider $\mathbf{heap} \in \mathcal{PV}$ to implicitly occur in field access expressions $\mathbf{o}.f$, in array access expressions $\mathbf{a}[\mathbf{i}]$, and in method calls $\mathbf{o}.m(\dots)$.

Proposition 2 (Non-occurring function and predicate symbols). *For all Kripke structures $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ and $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ differing only in the interpretation functions I vs. I' , all states $s \in \mathcal{S}$, all variable assignments β , and all $t \in Trm_{\Sigma} \cup Fma_{\Sigma} \cup Upd_{\Sigma}$: if for all function and predicate symbols $f \in \mathcal{F} \cup \mathcal{P}$ that syntactically occur in t we have $I(f) = I'(f)$, then we also have $val_{\mathcal{K},s,\beta}(t) = val_{\mathcal{K}',s,\beta}(t)$.*

Proposition 3 (Overwritten program variables). *For all Kripke structures \mathcal{K} , all states $s, s' \in \mathcal{S}$, all variable assignments β , all updates $(\mathbf{a} := t') \in Upd_{\Sigma}$*

where \mathbf{a} does not occur in t' , all $t \in \text{Trm}_\Sigma \cup \text{Fma}_\Sigma \cup \text{Upd}_\Sigma$, all $\varphi \in \text{Fma}_\Sigma$, and all program fragments \mathbf{p} : if for all program variables $\mathbf{b} \in \mathcal{PV} \setminus \{\mathbf{a}\}$ which occur in t or φ we have $s(\mathbf{b}) = s'(\mathbf{b})$, then we also have:

$$\begin{aligned} \text{val}_{\mathcal{K},s,\beta}(\{\mathbf{a} := t'\}t) &= \text{val}_{\mathcal{K},s',\beta}(\{\mathbf{a} := t'\}t) \\ \text{val}_{\mathcal{K},s,\beta}([\mathbf{a} = t'; \mathbf{p}]\varphi) &= \text{val}_{\mathcal{K},s',\beta}([\mathbf{a} = t'; \mathbf{p}]\varphi) \\ \text{val}_{\mathcal{K},s,\beta}(\langle \mathbf{a} = t'; \mathbf{p} \rangle \varphi) &= \text{val}_{\mathcal{K},s',\beta}(\langle \mathbf{a} = t'; \mathbf{p} \rangle \varphi) \end{aligned}$$

Prop. 3 holds because the initial value of the program variable \mathbf{a} is overwritten by the preceding update or assignment, and thus cannot influence the evaluation of t or φ , respectively.

Proposition 4 (Method calls). *Let \mathbf{p} be a method call statement ($\text{res} = \text{this.m}(\mathbf{p}_1, \dots, \mathbf{p}_n)$); let $\mathbf{hPre} : \text{Heap} \in \mathcal{PV}$, let $\text{reachableState} \in \text{Fma}_\Sigma$ be as in Def. 3 of [2], let $\text{reachableState}' \in \text{Fma}_\Sigma$ be as in Def. 4, and let $\text{noDeallocs} \in \text{Fma}_\Sigma$ be as in Def. 7. Then the following holds:*

$$\models \text{reachableState} \rightarrow \{\mathbf{hPre} := \mathbf{heap}\}[\mathbf{p}](\text{reachableState}' \wedge \text{noDeallocs})$$

Prop. 4 is guaranteed by the semantics of Java.

2 Proofs

2.1 Preparation

Lemma 1 (Relation between frame and anon). *Let $\text{mod} \in \text{Trm}_\Sigma^{\text{LocSet}}$, $\mathbf{hPre} : \text{Heap} \in \mathcal{PV}$, $\text{frame} \in \text{Fma}_\Sigma$ be as in Def. 3 of [2], $\text{noDeallocs} \in \text{Fma}_\Sigma$ be as in Def. 7, and let $\text{frame}' \in \text{Fma}_\Sigma$ be the formula*

$$\text{heap} \doteq \text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\}\text{mod}, \mathbf{heap}).$$

Then the following holds:

$$\models (\text{frame} \wedge \text{noDeallocs}) \leftrightarrow \text{frame}'$$

Proof. Let \mathcal{K} be a Kripke structure, $s \in \mathcal{S}$ be a state, β be a variable assignment, $h = s(\mathbf{heap})$, $h' = \text{val}_{\mathcal{K},s,\beta}(\text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\}\text{mod}, \mathbf{heap}))$, $s^{\text{pre}} = \text{val}_{\mathcal{K},s,\beta}(\mathbf{heap} := \mathbf{hPre})(s)$, $h^{\text{pre}} = s^{\text{pre}}(\mathbf{heap})$, $m^{\text{pre}} = \text{val}_{\mathcal{K},s^{\text{pre}},\beta}(\text{mod})$, $fl = I(\text{freshLocs})(h)$, and $fl^{\text{pre}} = I(\text{freshLocs})(h^{\text{pre}})$. Note that $h^{\text{pre}} = s(\mathbf{hPre})$. By definition of $I(\text{anon})$, we know that the following holds for all $o \in \mathcal{D}^{\text{Object}}$, $f \in \mathcal{D}^{\text{Field}}$:

$$h'(o, f) = \begin{cases} h(o, f) & \text{if } ((o, f) \in m^{\text{pre}} \text{ and } f \neq I(\text{created})) \\ & \text{or } (o, f) \in fl^{\text{pre}} \\ h^{\text{pre}}(o, f) & \text{otherwise} \end{cases} \quad (1)$$

We first show that $(\mathcal{K}, s, \beta) \models \text{frame} \wedge \text{noDeallocs}$ implies that $(\mathcal{K}, s, \beta) \models \text{frame}'$, and then the other way round.

1. Let $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$. Using the definitions of *frame*, *noDeallocs* and *frame'*, we assume

$$(o, f) \in m^{pre} \cup fl^{pre} \text{ or } h(o, f) = h^{pre}(o, f) \quad (2)$$

$$\text{if } (o, f) \in fl, \text{ then } (o, f) \in fl^{pre} \quad (3)$$

$$h(I(\mathbf{null}), I(created)) = h^{pre}(I(\mathbf{null}), I(created)) \quad (4)$$

and aim to show

$$h'(o, f) = h(o, f). \quad (5)$$

From (2) we get that one of the following three cases must apply:

- $(o, f) \in m^{pre}$. If $f \neq I(created)$ or $(o, f) \in fl^{pre}$, then (5) immediately follows from (1). We thus assume

$$f = I(created) \quad (6)$$

$$(o, f) \notin fl^{pre}. \quad (7)$$

Now, (1) yields

$$h'(o, f) = h^{pre}(o, f). \quad (8)$$

If $o = I(\mathbf{null})$, then we get from (4) that $h(o, f) = h^{pre}(o, f)$, which together with (8) immediately yields (5). Thus we assume

$$o \neq I(\mathbf{null}). \quad (9)$$

From (3) and (7) we get that

$$(o, f) \notin fl.$$

This, (9), and the definition of $I(\text{freshLocs})$ imply $h(o, I(created)) = tt$. Analogously, (7) and (9) imply $h^{pre}(o, I(created)) = tt$. Together, we have $h(o, I(created)) = h^{pre}(o, I(created))$, which because of (6) can be written as $h(o, f) = h^{pre}(o, f)$. We combine this with (8) to get (5).

- $(o, f) \in fl^{pre}$. Then (1) immediately yields (5).
 - $h(o, f) = h^{pre}(o, f)$. If $(o, f) \in m^{pre}$ or $(o, f) \in fl^{pre}$, then the proof proceeds as for the respective case above. Otherwise, (1) guarantees that $h'(o, f) = h^{pre}(o, f)$, and thus we have (5).
2. Let $o \in \mathcal{D}^{Object}$, $f \in \mathcal{D}^{Field}$. We assume (5), and show first (2), then (3), and finally (4).
 - (a) If $(o, f) \in m^{pre}$ or $(o, f) \in fl^{pre}$, then (2) holds trivially. Otherwise, (5) and (1) imply $h(o, f) = h^{pre}(o, f)$, which also implies (2).
 - (b) We prove (3) by contradiction: we assume that $(o, f) \in fl \setminus fl^{pre}$. By definition of $I(\text{freshLocs})$, this means that $o \neq I(\mathbf{null})$, that $h(o, I(created)) = ff$, and that $h^{pre}(o, I(created)) = tt$. From (5) and (1) we get that $h(o, I(created)) = h^{pre}(o, I(created))$. Together, we have $ff = tt$.
 - (c) The definition of $I(\text{freshLocs})$ tells us that $(I(\mathbf{null}), I(created)) \notin fl^{pre}$. Thus, (5) and (1) immediately guarantee (4). \square

2.2 Method Contracts

Theorem 1 (Soundness of useMethodContract). *Let $\Gamma, \Delta \in 2^{Fma_\Sigma}$, $u \in Upd_\Sigma$, $\llbracket \cdot \rrbracket \in \{\llbracket \cdot \rrbracket, \langle \cdot \rangle\}$, $\mathbf{r} \in \mathcal{PV}$, $\mathbf{o} \in Trm_\Sigma$, the method \mathbf{m} , $\mathbf{p}'_1, \dots, \mathbf{p}'_n \in Trm_\Sigma$, $\varphi \in Fma_\Sigma$, $A \in \mathcal{T}$, $mct = (\mathbf{m}, \mathbf{this}, (\mathbf{p}_1, \dots, \mathbf{p}_n), \mathbf{res}, \mathbf{hPre}, pre, post, mod, \tau)$, $reachableState, reachableState' \in Fma_\Sigma$, $v, w \in Upd_\Sigma$, and $h, r' \in \mathcal{F}$ all be as in Def. 4 of [2]. If*

$$\models \Gamma \Rightarrow \{u\}\{w\}(pre \wedge reachableState), \Delta \quad (10)$$

$$\models \Gamma \Rightarrow \{u\}\{w\}\{\mathbf{hPre} := \mathbf{heap}\}\{v\}(post \wedge reachableState' \rightarrow \llbracket \dots \rrbracket \varphi), \Delta \quad (11)$$

and if for all types $B \sqsubseteq A$ we have

$$\models CorrectMethodContract(mct, B), \quad (12)$$

then the following holds:

$$\models \Gamma \Rightarrow \{u\}\llbracket \mathbf{r} = \mathbf{o.m}(\mathbf{p}'_1, \dots, \mathbf{p}'_n); \dots \rrbracket \varphi, \Delta.$$

Proof. Let (10), (11) and (12) hold. Let furthermore $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ be a Kripke structure, $s \in \mathcal{S}$, and β be a variable assignment. Our goal is to show

$$(\mathcal{K}, s, \beta) \models \Gamma \Rightarrow \{u\}\llbracket \mathbf{r} = \mathbf{o.m}(\mathbf{p}'_1, \dots, \mathbf{p}'_n); \dots \rrbracket \varphi, \Delta.$$

If there is $\gamma \in \Gamma$ with $val_{\mathcal{K}, s, \beta}(\gamma) = \text{ff}$ or if there is $\delta \in \Delta$ with $val_{\mathcal{K}, s, \beta}(\delta) = \text{tt}$, then this is trivially true. We therefore assume that

$$(\mathcal{K}, s, \beta) \models \bigwedge (\Gamma \cup \neg \Delta), \quad (13)$$

and aim to show that $(\mathcal{K}, s, \beta) \models \{u\}\llbracket \mathbf{r} = \mathbf{o.m}(\mathbf{p}'_1, \dots, \mathbf{p}'_n); \dots \rrbracket \varphi$.

Let $s_1 = val_{\mathcal{K}, s, \beta}(u)(s)$. Then our goal is to show

$$(\mathcal{K}, s_1, \beta) \models \llbracket \mathbf{r} = \mathbf{o.m}(\mathbf{p}'_1, \dots, \mathbf{p}'_n); \dots \rrbracket \varphi.$$

Let $s_2 = val_{\mathcal{K}, s_1, \beta}(w)(s_1)$. Because of the definition of w , it holds for all $\mathbf{a} \in \mathcal{PV} \setminus \{\mathbf{this}, \mathbf{p}_1, \dots, \mathbf{p}_n\}$ that $s_1(\mathbf{a}) = s_2(\mathbf{a})$. Since by Def. 4 neither \mathbf{this} nor $\mathbf{p}_1, \dots, \mathbf{p}_n$ occur in the above formula, Prop. 1 tells us that the interpretation of this formula is the same in s_1 and s_2 . It is therefore sufficient if we show

$$(\mathcal{K}, s_2, \beta) \models \llbracket \mathbf{r} = \mathbf{o.m}(\mathbf{p}'_1, \dots, \mathbf{p}'_n); \dots \rrbracket \varphi.$$

The definition of w and Prop. 1 ensure that $s_2(\mathbf{this}) = val_{\mathcal{K}, s_2, \beta}(\mathbf{o})$, and that $s_2(\mathbf{p}_1) = val_{\mathcal{K}, s_2, \beta}(\mathbf{p}'_1)$, \dots , $s_2(\mathbf{p}_n) = val_{\mathcal{K}, s_2, \beta}(\mathbf{p}'_n)$. Thus, we can aim to prove the formula below instead of the formula above:

$$(\mathcal{K}, s_2, \beta) \models \llbracket \mathbf{r} = \mathbf{this.m}(\mathbf{p}_1, \dots, \mathbf{p}_n); \dots \rrbracket \varphi.$$

Since by Def. 4 the program variable \mathbf{res} does not occur in the above formula, the Java semantics allows us to instead show

$$(\mathcal{K}, s_2, \beta) \models \llbracket \mathbf{res} = \mathbf{this.m}(\mathbf{p}_1, \dots, \mathbf{p}_n); \mathbf{r} = \mathbf{res}; \dots \rrbracket \varphi.$$

Let $s_3 = \text{val}_{\mathcal{K}, s_2, \beta}(\mathbf{hPre} := \mathbf{heap})(s_2)$. Since by Def. 4 the program variable \mathbf{hPre} does not occur in the above formula, by Prop. 1 it is sufficient if we prove

$$(\mathcal{K}, s_3, \beta) \models \llbracket \mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n); \mathbf{r} = \mathbf{res}; \dots \rrbracket \varphi. \quad (\text{thm1-goal})$$

We combine (13) with (10) to get

$$(\mathcal{K}, s, \beta) \models \{u\}\{w\}(pre \wedge \text{reachableState}),$$

which by definition of s_2 is the same as

$$(\mathcal{K}, s_2, \beta) \models pre \wedge \text{reachableState}. \quad (14)$$

Let $C = \delta(s_2(\mathbf{this}))$. This means that

$$(\mathcal{K}, s_2, \beta) \models \text{exactInstance}_C(\mathbf{this}). \quad (15)$$

Since $\alpha(\mathbf{this}) = A$, we have $C \sqsubseteq A$ because of well-typedness. Instantiating (12) with C and s_2 yields

$$\begin{aligned} (\mathcal{K}, s_2, \beta) &\models pre \wedge \text{reachableState} \wedge \text{exactInstance}_C(\mathbf{this}) \\ &\rightarrow \{\mathbf{hPre} := \mathbf{heap}\} \llbracket \mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n); \rrbracket' (post \wedge \text{frame}) \end{aligned}$$

where $\llbracket \cdot \rrbracket'$ is $\langle \cdot \rangle$ if $\llbracket \cdot \rrbracket$ is $\langle \cdot \rangle$, and where $\llbracket \cdot \rrbracket'$ is either $\langle \cdot \rangle$ or $[\cdot]$ otherwise. Together with (14) and (15), this implies

$$(\mathcal{K}, s_2, \beta) \models \{\mathbf{hPre} := \mathbf{heap}\} \llbracket \mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n); \rrbracket' (post \wedge \text{frame}).$$

With the definition of s_3 , this becomes

$$(\mathcal{K}, s_3, \beta) \models \llbracket \mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n); \rrbracket' (post \wedge \text{frame}). \quad (16)$$

If there is no $s_4 \in \mathcal{S}$ such that $(s_3, s_4) \in \rho(\mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n);)$ (i.e., if the method call does not terminate when started in s_3), then (16) implies that $\llbracket \cdot \rrbracket'$ must be $[\cdot]$, and thus $\llbracket \cdot \rrbracket$ also must be $[\cdot]$. Then, (thm1-goal) holds trivially, because there is no final state which would have to satisfy φ .

We can thus find $s_4 \in \mathcal{S}$ such that $(s_3, s_4) \in \rho(\mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n);)$. As our programs are deterministic, s_4 is the only such state. Our proof goal (thm1-goal) now becomes

$$(\mathcal{K}, s_4, \beta) \models \llbracket \mathbf{r} = \mathbf{res}; \dots \rrbracket \varphi. \quad (\text{thm1-goal}')$$

From (16) and the definition of s_4 we get

$$(\mathcal{K}, s_4, \beta) \models post \wedge \text{frame}. \quad (17)$$

Let $\text{noDeallocs} \in \text{Fma}_{\Sigma}$ be as in Def. 7. Prop. 4 tells us that

$$\begin{aligned} (\mathcal{K}, s_2, \beta) &\models \text{reachableState} \\ &\rightarrow \{\mathbf{hPre} := \mathbf{heap}\} \llbracket \mathbf{res} = \mathbf{this.m}(p_1, \dots, p_n); \rrbracket \\ &\quad (\text{reachableState}' \wedge \text{noDeallocs}). \end{aligned}$$

Together with (14) and the definition of s_4 , this turns into

$$(\mathcal{K}, s_4, \beta) \models \text{reachableState}' \wedge \text{noDeallocs}. \quad (18)$$

Let $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ be a Kripke structure identical to \mathcal{K} , except that $I'(h) = s_4(\mathbf{heap})$, and except that $I'(r') = s_4(\mathbf{res})$. Since by Def. 4 the symbols h and r' do not occur in Γ nor in Δ , we get from (13) that $(\mathcal{K}', s, \beta) \models \bigwedge(\Gamma \cup \neg\Delta)$. This and (11) imply

$$(\mathcal{K}', s, \beta) \models \{u\}\{w\}\{\mathbf{hPre} := \mathbf{heap}\}\{v\}(\text{post} \wedge \text{reachableState}' \rightarrow \llbracket \dots \rrbracket \varphi).$$

As h and r' do not occur in u , in w or in $\mathbf{hPre} := \mathbf{heap}$, the above and Prop. 2 imply that

$$(\mathcal{K}', s_3, \beta) \models \{v\}(\text{post} \wedge \text{reachableState}' \rightarrow \llbracket \dots \rrbracket \varphi).$$

Let $s'_4 = \text{val}_{\mathcal{K}', s_3, \beta}(v)(s_3)$. Then the above implies

$$(\mathcal{K}', s'_4, \beta) \models \text{post} \wedge \text{reachableState}' \rightarrow \llbracket \dots \rrbracket \varphi.$$

Since h and r' do not occur in the above formula, by Prop. 2 we get that

$$(\mathcal{K}, s'_4, \beta) \models \text{post} \wedge \text{reachableState}' \rightarrow \llbracket \dots \rrbracket \varphi. \quad (19)$$

Given the definition of s_4 , the semantics of Java tells us that for all $a \in \mathcal{PV} \setminus \{\mathbf{heap}, \mathbf{res}\}$ we have $s_3(\mathbf{a}) = s_4(\mathbf{a})$. Similarly, the definition of s'_4 implies that for all $\mathbf{a} \in \mathcal{PV} \setminus \{\mathbf{heap}, \mathbf{r}, \mathbf{res}\}$ we have $s_3(\mathbf{a}) = s'_4(\mathbf{a})$. Together, we have

$$\text{for all } \mathbf{a} \in \mathcal{PV} \setminus \{\mathbf{heap}, \mathbf{r}, \mathbf{res}\} : s'_4(\mathbf{a}) = s_4(\mathbf{a}). \quad (20)$$

The definition of s'_4 also guarantees that

$$s'_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_3, \beta}(\text{anon}(\mathbf{heap}, \text{mod}, h)) \quad (21)$$

$$s'_4(\mathbf{r}) = I'(r') = s_4(\mathbf{res}) \quad (22)$$

$$s'_4(\mathbf{res}) = I'(r') = s_4(\mathbf{res}) \quad (23)$$

Using (17) and (18), Lemma 1 tells us that

$$(\mathcal{K}, s_4, \beta) \models \mathbf{heap} \doteq \text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} \text{mod}, \mathbf{heap}),$$

which we can also express as

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}, s_4, \beta}(\text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} \text{mod}, \mathbf{heap})).$$

Since by Def. 4 the function symbols h and r' do not occur in the above formula, and since \mathcal{K}' is otherwise identical to \mathcal{K} , Prop. 2 yields

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_4, \beta}(\text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} \text{mod}, \mathbf{heap})).$$

As we defined \mathcal{K}' such that $I'(h) = s_4(\mathbf{heap})$, this implies

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_4, \beta}(\text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} \text{mod}, h)).$$

Since s_3 and s_4 are identical except for \mathbf{heap} and \mathbf{res} , and since \mathbf{res} does not occur in $\{\mathbf{heap} := \mathbf{hPre}\} \text{mod}$, Prop. 3 tells us that $\text{val}_{\mathcal{K}, s_4, \beta}(\{\mathbf{heap} := \mathbf{hPre}\} \text{mod}) = \text{val}_{\mathcal{K}, s_3, \beta}(\{\mathbf{heap} := \mathbf{hPre}\} \text{mod})$. As \mathbf{heap} and \mathbf{res} do not occur in the other arguments of anon , we can transform the statement above into

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_3, \beta}(\text{anon}(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} \text{mod}, h)).$$

The definition of s_3 implies $s_3(\mathbf{heap}) = s_3(\mathbf{hPre})$. Thus, the update $\mathbf{heap} := \mathbf{hPre}$ has no effect in s_3 . This allows simplifying the above into

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_3, \beta}(\text{anon}(\mathbf{hPre}, \text{mod}, h)),$$

and replacing \mathbf{hPre} with \mathbf{heap} to get

$$s_4(\mathbf{heap}) = \text{val}_{\mathcal{K}', s_3, \beta}(\text{anon}(\mathbf{heap}, \text{mod}, h)).$$

This, together with (21), implies that $s_4(\mathbf{heap}) = s'_4(\mathbf{heap})$. Combining this result with (20) and (23) yields that s_4 and s'_4 differ at most in \mathbf{r} . Since by Def. 4 the program variable \mathbf{r} does not occur in post , (17) and Prop. 1 imply

$$(\mathcal{K}, s'_4, \beta) \models \text{post}. \quad (24)$$

As \mathbf{r} also does not occur in $\text{reachableState}'$, we get from (18) that

$$(\mathcal{K}, s'_4, \beta) \models \text{reachableState}'.$$

This, (24) and (19) together imply

$$(\mathcal{K}, s'_4, \beta) \models \llbracket \dots \rrbracket \varphi.$$

By (22) and (23), we know that $s'_4(\mathbf{res}) = s'_4(\mathbf{r})$. Thus, the Java semantics allows us to rewrite the above statement into

$$(\mathcal{K}, s'_4, \beta) \models \llbracket \mathbf{r} = \mathbf{res}; \dots \rrbracket \varphi.$$

Finally, as s_4 and s'_4 differ at most in \mathbf{r} , Prop. 3 tells us that

$$(\mathcal{K}, s_4, \beta) \models \llbracket \mathbf{r} = \mathbf{res}; \dots \rrbracket \varphi,$$

and this is property (thm1-goal') which we aimed to show. \square

2.3 Dependency Contracts

Theorem 2 (Soundness of useDependencyContract). *Let $\Gamma, \Delta \in 2^{Fma_\Sigma}$, $obs \in \mathcal{F} \cup \mathcal{P}$, $h^{new} = (f_1(f_2(\dots(f_m(h^{base}, \dots))))))$, $\circ, \mathbf{p}'_1, \dots, \mathbf{p}'_n \in \text{Trm}_\Sigma$, $A \in \mathcal{T}$, $\text{depct} = (obs, \mathbf{this}, (\mathbf{p}_1, \dots, \mathbf{p}_n), \text{pre}, \text{dep})$, $\mathbf{hPre} \in \mathcal{PV}$, $\text{mod} = \text{allLocs} \setminus \text{dep}$,*

$reachableState, frame, noDeallocs \in Fma_{\Sigma}$, $w \in Upd_{\Sigma}$, $guard, equal \in Fma_{\Sigma}$ all be as in Def. 7 of [2]. If

$$\models \Gamma, guard \rightarrow equal \Rightarrow \Delta \quad (25)$$

and if for all types $B \sqsubseteq A$ we have

$$\models CorrectDependencyContract(depct, B), \quad (26)$$

then the following holds:

$$\models \Gamma \Rightarrow \Delta.$$

Proof. Let (25) and (26) hold, and let $\mathcal{K} = (\mathcal{D}, \delta, I, \mathcal{S}, \rho)$ be a Kripke structure. Our goal is to show $(\mathcal{K}, s, \beta) \models \Gamma \Rightarrow \Delta$. We will do a proof by contradiction and assume that this does *not* hold, or in other words, that $(\mathcal{K}, s, \beta) \models \bigwedge(\Gamma \cup \neg\Delta)$ holds. This and (25) imply $(\mathcal{K}, s, \beta) \models \neg(guard \rightarrow equal)$, which means that $(\mathcal{K}, s, \beta) \models guard \wedge \neg equal$. If we insert the definitions of $guard$ and $equal$, and distribute the update w over the conjuncts of $guard$, then this reads as

$$\begin{aligned} (\mathcal{K}, s, \beta) &\models \{w\}\{\mathbf{heap} := h^{base}\}(pre \wedge reachableState) \\ (\mathcal{K}, s, \beta) &\models \{w\}\{\mathbf{hPre} := h^{base} \parallel \mathbf{heap} := h^{new}\}(frame \wedge noDeallocs) \\ (\mathcal{K}, s, \beta) &\models \neg(obs(h^{new}, \mathbf{o}, \mathbf{p}'_1, \dots, \mathbf{p}'_n) \equiv obs(h^{base}, \mathbf{o}, \mathbf{p}'_1, \dots, \mathbf{p}'_n)) \end{aligned} \quad (27)$$

Let $s_1 = val_{\mathcal{K}, s, \beta}(w)(s)$. Then the first two statements above become

$$\begin{aligned} (\mathcal{K}, s_1, \beta) &\models \{\mathbf{heap} := h^{base}\}(pre \wedge reachableState) \\ (\mathcal{K}, s_1, \beta) &\models \{\mathbf{hPre} := h^{base} \parallel \mathbf{heap} := h^{new}\}(frame \wedge noDeallocs) \end{aligned}$$

Let $s_1^{base} = val_{\mathcal{K}, s, \beta}(\mathbf{heap} := h^{base})(s_1)$, $s_1^{new} = val_{\mathcal{K}, s, \beta}(\mathbf{hPre} := h^{base} \parallel \mathbf{heap} := h^{new})(s_1)$. Then the statements above turn into

$$(\mathcal{K}, s_1^{base}, \beta) \models pre \wedge reachableState \quad (28)$$

$$(\mathcal{K}, s_1^{new}, \beta) \models frame \wedge noDeallocs \quad (29)$$

As $\mathbf{this}, \mathbf{p}_1, \dots, \mathbf{p}_n$ do not occur in (27), and as s and s_1 are otherwise identical, we get by Prop. 1 that

$$(\mathcal{K}, s_1, \beta) \models \neg(obs(h^{new}, \mathbf{o}, \mathbf{p}'_1, \dots, \mathbf{p}'_n) \equiv obs(h^{base}, \mathbf{o}, \mathbf{p}'_1, \dots, \mathbf{p}'_n)),$$

which because of the definition of s_1 implies that

$$(\mathcal{K}, s_1, \beta) \models \neg(obs(h^{new}, \mathbf{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \equiv obs(h^{base}, \mathbf{this}, \mathbf{p}_1, \dots, \mathbf{p}_n)). \quad (30)$$

Lemma 1 and (29) tell us that

$$(\mathcal{K}, s_1^{new}, \beta) \models \mathbf{heap} \doteq anon(\mathbf{hPre}, \{\mathbf{heap} := \mathbf{hPre}\} mod, \mathbf{heap}),$$

which because of the definition of s_1^{new} is the same as

$$(\mathcal{K}, s_1, \beta) \models h^{new} \doteq \text{anon}(h^{base}, \{\text{heap} := h^{base}\} \text{mod}, h^{new}). \quad (31)$$

Let $C = \delta(s_1^{base}(\text{this}))$. This means that

$$(\mathcal{K}, s_1^{base}, \beta) \models \text{exactInstance}_C(\text{this}). \quad (32)$$

Let $\mathcal{K}' = (\mathcal{D}, \delta, I', \mathcal{S}, \rho)$ be a Kripke structure identical to \mathcal{K} , except that $I'(h) = \text{val}_{\mathcal{K}, s_1, \beta}(h^{new})$. Since $\alpha(\text{this}) = A$, we have $C \sqsubseteq A$. Instantiating (26) with C , \mathcal{K}' and s_1^{base} yields

$$\begin{aligned} (\mathcal{K}', s_1^{base}, \beta) &\models \text{pre} \wedge \text{reachableState} \wedge \text{exactInstance}_C(\text{this}) \\ &\rightarrow \text{obs}(\text{heap}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \{\text{heap} := \text{anon}(\text{heap}, \text{mod}, h)\} \\ &\quad \text{obs}(\text{heap}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

As h does not occur in (28) or (32), we have $(\mathcal{K}', s_1^{base}, \beta) \models \text{pre} \wedge \text{reachableState} \wedge \text{exactInstance}_C(\text{this})$ by Prop. 2, which we can combine with the statement above to get

$$\begin{aligned} (\mathcal{K}', s_1^{base}, \beta) &\models \text{obs}(\text{heap}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \{\text{heap} := \text{anon}(\text{heap}, \text{mod}, h)\} \text{obs}(\text{heap}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

Applying the update yields

$$\begin{aligned} (\mathcal{K}', s_1^{base}, \beta) &\models \text{obs}(\text{heap}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \text{obs}(\text{anon}(\text{heap}, \text{mod}, h), \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

Because of the definition of s_1^{base} , this is the same as

$$\begin{aligned} (\mathcal{K}', s_1, \beta) &\models \text{obs}(h^{base}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \text{obs}(\text{anon}(h^{base}, \{\text{heap} := h^{base}\} \text{mod}, h), \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

By definition of \mathcal{K}' , we have $I'(h) = \text{val}_{\mathcal{K}, s_1, \beta}(h^{new})$. As h does not occur in h^{new} , and as \mathcal{K} and \mathcal{K}' are otherwise identical, Prop. 2 guarantees that $\text{val}_{\mathcal{K}, s_1, \beta}(h^{new}) = \text{val}_{\mathcal{K}', s_1, \beta}(h^{new})$. Thus, we have $I'(h) = \text{val}_{\mathcal{K}', s_1, \beta}(h^{new})$, and can thus write the statement above as

$$\begin{aligned} (\mathcal{K}', s_1, \beta) &\models \text{obs}(h^{base}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \text{obs}(\text{anon}(h^{base}, \{\text{heap} := h^{base}\} \text{mod}, h^{new}), \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

As the function symbol h does not occur in the above formula, and as \mathcal{K} and \mathcal{K}' are otherwise identical, Prop. 2 tells us that

$$\begin{aligned} (\mathcal{K}, s_1, \beta) &\models \text{obs}(h^{base}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \\ &\equiv \text{obs}(\text{anon}(h^{base}, \{\text{heap} := h^{base}\} \text{mod}, h^{new}), \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n). \end{aligned}$$

We can combine this with (31) to get

$$(\mathcal{K}, s_1, \beta) \models \text{obs}(h^{base}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n) \equiv \text{obs}(h^{new}, \text{this}, \mathbf{p}_1, \dots, \mathbf{p}_n),$$

which contradicts (30). \square

References

1. J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Second Edition*. Addison-Wesley, 2000.
2. P. H. Schmitt, M. Ulbrich, and B. Weiß. Dynamic frames in Java dynamic logic. In B. Beckert and C. Marché, editors, *Proceedings, International Conference on Formal Verification of Object-Oriented Software (FoVeOOS 2010)*, LNCS. Springer, 2010. To appear.